The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 6, pp.33-39, Dec. 31, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.6.33
JIIBC 2020-6-6

Gradle 빌드 오류 해결을 위한 솔루션 추천 방안

An Approach to Recommending of Solutions for Resolving Gradle Build Error

강민구*, 김태영*, 김순태**, 류덕산**

Mingu Kang*, Taeyoung Kim*, Suntae Kim**, Duksan Ryu**

요 약 개발자는 프로젝트 구축과정에서 빌드 되지 않는 코드를 수동으로 복구하는데 상당한 시간을 소비한다. 빌드가 실패하면 실패한 실행을 이해하고 실패 원인을 식별한 뒤, 솔루션을 구현하는 과정이 필요하다. 이러한 노력을 줄이고 프로젝트 구축을 자동화하기 위해 Gradle과 같은 빌드 도구들이 발전되어왔다. 하지만 빌드 도구들은 여전히 많은 오류를 해결하지 못하여 개발자들에게 빌드 오류를 해결하기 위한 노력을 필요로 한다. 본 연구에서는 Gradle 빌드의 성공률을 높이고 오류 해결에 필요한 노력을 줄이기 위한 솔루션 추천 방안을 제시한다. 우리는 빌드 오류를 수집하는 방법과 빌드 오류 메시지로부터 성공적인 빌드로 전환되도록 하는 방법을 제공한다. 특히 Github의 Java 프로젝트에서 수집한 296개의 빌드 오류 메시지를 솔루션으로 분류하고 89%가 솔루션이 적용될 수 있음을 보여준다.

Abstract Developers spend considerable time manually repairing code that was not built during project construction. If the build fails, it is necessary to understand the failed execution, identify the cause of the failure, and then implement the solution. Build tools such as Gradle have been developed to reduce this effort and automate project construction. However, build tools still do not solve many errors, requiring developers to try to solve build errors. In this study, we propose a solution recommendation method to increase the success rate of Gradle build and reduce the effort required to resolve errors. We provide a way to collect build errors and a way to transition from build error messages to successful builds. In particular, 296 build error messages collected from Github's Java project are classified as solutions, and 89% show that the solution can be applied.

Key Words : Build Error, Build Tools, Gradle, Java, Machine Learning

Ⅰ. 서 론

Github와 같은 Version Control System(VCS)은 소프트웨어 개발 실무자들에게 인기를 얻으면서 배포되는 소프트웨어의 수가 증가하고 있다. VCS에 배포된 소프트웨어를 이용하는 개발자들의 목표는 대상 소프트웨어를 자신의 환경으로 올바르게 빌드하는 것이다. 성공적으로 빌드가 수행되면 소프트웨어가 이용자의 환경에서도 올바르게 동작한다. 개발자들은 이러한 동작을 확인하기 위해서 대상 소프트웨어를 수동적으로 컴파일, 테스

*준회원, 전북대학교 소프트웨어공학과
*정회원, 전북대학교 소프트웨어공학과
접수일자 2020년 10월 26일, 수정완료 2020년 11월 26일
게재확정일자 2020년 12월 4일

Received: 26 October, 2020 / Revised: 26 November, 2020 / Accepted: 4 December, 2020
*Corresponding Author: stkim@jbnu.ac.kr
Department of Software Engineering, CAIIT, Jeonbuk National University, South Korea

팅, 및 실행을 수행한다. 이 과정에서 빌드에 실패하면 개발자들은 오류를 해결하기 위해 빌드에 실패한 실행을 이해하고 오류를 발생시키는 원인을 식별해야 한다. 이러한 수동적인 작업은 개발자들에게 많은 시간과 비용을 발생시키므로 자동화된 빌드 기술이 중요하다.

하지만 기존의 Gradle, Maven 및 Ant 와 같은 자동화된 빌드 도구들은 많은 발전에도 불구하고 본 연구에서 조사 결과 여전히 프로젝트 빌드는 평균적으로 46% 정도 실패하게 되어 오류 해결을 위한 노력을 필요로 하고 있다. 이러한 빌드 오류들은 소프트웨어가 개발되었던 환경이 아닌, 다른 환경에서 존재하지 않는 구성요소에 의존할 때 발생한다[1]. 발생한 오류를 해결하기 위해 개발자들은 하나 이상의 솔루션을 적용해 수정을 확인하는 과정이 필요하다[2][3]. 특히, 오류 메시지를 읽는 노력은 오류 해결 작업의 25%까지 할당될 수 있다[4].

본 연구에서는 빌드의 성공률을 높이고 오류 해결에 필요한 노력을 줄이기 위해 Gradle이 빌드하지 못하는 오류의 해결을 위한 솔루션 추천 연구를 수행한다. 먼저, Github 저장소에서 빌드 도구인 Gradle로 빌드된 Java 프로젝트를 수집한다. 다음으로, Gradle의 기본 빌드 명령어를 통해 수집한 프로젝트들을 빌드한다. 빌드에 실패한 프로젝트들을 분석하여 오류 원인을 식별하고, 성공적인 빌드를 위한 솔루션 범주를 수동적으로 분류한다. 마지막으로, 빌드 오류에 대한 해결을 위해 오류 메시지와 관련된 솔루션 추천 모델을 구축한다.

우리는 Github 저장소에서 수집한 프로젝트들로부터 빌드에 실패한 200개의 프로젝트와 296개의 오류 메시지에 대한 실험을 수행하였다. 실험 결과는 높은 Accuracy를 통해 14가지로 범주화된 솔루션에 대해 관련된 오류 메시지를 성공적으로 추천해 줄 수 있음을 보여준다. 실험에는 SVM(Support vector machine), Logistic Regression, Random Forest, XGBoost 알고리즘을 비교하였고, Random Forest의 경우 89.52%의 가장 높은 성능을 보여준다. 우리의 연구는 빌드 실패의 원인을 분석하고 실패에 대한 솔루션을 제공하므로 오류를 식별하고 수정하는데 필요한 노력을 감소시킨다.

II. 관련 연구

이 절에서는 프로젝트 구축과정에서 발생하는 오류를 분석하고 이를 해결하고자 하는 연구들을 소개한다.

Kim[5]은 OSEK/VDX 운영체제를 지원하기 위한 통합개발환경을 구축하여 소스 코드의 빌드, 디버깅 및 자동 코드 생성기능을 지원한다. Lee[6]는 다양한 플랫폼에 대한 노드 소프트웨어를 효율적으로 개발할 수 있는 소프트웨어 개발 지원 도구를 제안한다. 제안 시스템은 코드 템플릿을 바탕으로 소스 코드를 자동으로 생성할 수 있도록 해준다. 하지만 위 연구들은 시스템 동작 과정에서 발생하는 에러에 대한 해결책은 포함하고 있지 않다.

Mclntosh[7]는 대규모 오픈 소스 프로젝트에 대한 빌드 시스템의 복잡성을 분석하고 비용을 측정했다. 대규모 시스템에서 빌드 환경으로 인한 빌드 실패를 해결하기 위해 수정 작업을 여러 번 수행해야 함을 보여준다. Seo[8]는 2,660만 개의 컴파일러 오류에 관한 경험적 연구를 제공한다. 프로그램 개발 과정에서 컴파일러 오류가 생성되는 워크 플로우를 식별 및 설명하고 실패 빈도와 오류 유형을 분석한다.

최근 연구들은 프로젝트를 구축하기 위해 사용하는 자동화된 빌드 도구들의 오류에 중점을 두고 있다. Sulír[1]은 빌드 실패 원인과 그에 영향을 미치는 요인을 분석한다. 관찰된 요인으로 가장 빈번하게 발생하는 빌드 실패는 종속성 문제임을 보여준다. Hassan[9]은 오픈 소스 소프트웨어의 빌드 실패 유형을 계층적으로 분류하고 빌드 실패의 자동해결을 연구한다. 위 두 연구는 빌드 오류 메시지를 범주화하여 오류 유형을 분류한다. 하지만 빌드 오류 메시지가 항상 빌드 실패의 근본적인 원인을 나타내지는 않는다[8]. Hassan[10]은 Gradle 빌드 스크립트의 수정 패턴을 추출하여 빌드 복구 연구를 수행한다. 하지만 위 연구들은 자동적인 빌드 오류 해결을 위해 해결 가능한 오류를 수동적으로 식별해야 한다.

본 연구에서는 성공적인 빌드가 수행될 때까지 반복 수정하여 실패를 솔루션으로 범주화한다. 또한, 기계학습을 통하여 새로운 프로젝트 및 빌드 도구에 쉽게 확장 가능한, 자동화된 솔루션 추천 방법을 제안한다.

III. 연구 방법

이 절에서는 개발자가 프로젝트 구축과정에서 직면할 수 있는 빌드 오류를 해결하기 위한 솔루션 추천의 전체 개요를 보여준다. 그림 1에 나타나 있는 프로세스는 4 단계로 이루어져 있다. 첫 번째 단계에서 Github 저장소에서 연구 대상으로 사용할 프로젝트를 선택한다. 이 단계에서 연구 대상 프로젝트 선정 기준이 제시된다. 두 번째 단계에서 Gradle의 기본 빌드 명령어를 통해 빌드 실패

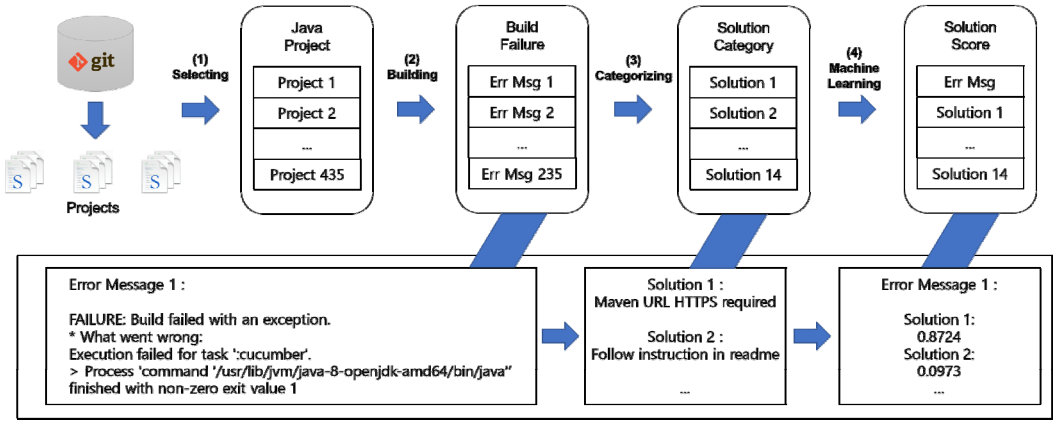


그림 1. 빌드 오류 해결을 위한 솔루션 추천의 전반적인 과정
Fig. 1. Overall process of recommending solutions for resolving build errors

프로젝트를 수집한다. 세 번째 단계에서는 빌드에 실패한 프로젝트들이 가지고 있는 오류들을 식별하고 빌드 성공을 이루는 솔루션으로 분류하여 범주화한다. 마지막으로, 빌드 오류 메시지와 분류된 범주를 이용하여 빌드 실패와 관련된 솔루션 추천을 위한 기계학습 모델을 구축한다.

1. 프로젝트 선택

개발자의 프로젝트 빌드 수행 과정에서 직면하는 빌드 실패를 분석하기 위해 Github에서 Gradle로 빌드된 프로젝트를 수집한다. 이 저장소에서 우리는 다음과 같은 프로젝트 선택 기준을 가지고 500개의 프로젝트를 정렬 순서대로 추출했다.

- "Gradle" 용어로 검색된 프로젝트
- Java로 작성된 프로젝트
- Android 또는 Java Micro Edition(ME)로 작성되지 않은 프로젝트
- 별의 수가 50개 이하인 프로젝트
- gradle wrapper를 포함하는 프로젝트

본 연구에서는 순수 Java 언어로 작성된 프로젝트에 집중적으로 분석하기 위해 Android 및 Java ME를 제외했다. 추가적으로, 검색에는 별의 개수가 50개 이하인 프로젝트만 검색되도록 쿼리를 추가했다. 상대적으로 별의 개수가 적은 프로젝트는 협력자가 적고 이슈에 대한 피드백의 수가 적어 이용자가 쉽게 오류에 직면할 수 있으므로 선택 기준으로 사용했다.

gradle wrapper(gradlew)는 새로운 환경에서 프로젝트를 설정할 때 Java나 gradle을 설치하지 않고 빌드할 수 있게 해주는 역할을 수행한다. gradlew를 포함하지 않는 프로젝트는 로컬 환경에 설치된 Java 버전과 gradle 버전에 영향을 받게 된다. 따라서, gradlew을 포함하지 않는 프로젝트들은 선택에서 제외된다. 결과적으로 프로젝트 수집 기준에 부합하는 프로젝트 셋은 435개로 구성된다.

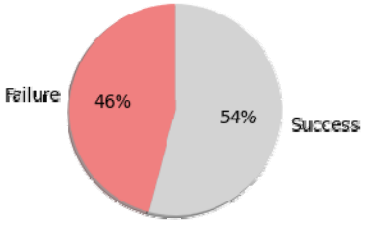


그림 2. 프로젝트 빌드 실패와 성공 비율
Fig. 2. Project Build Failure vs. Success ratio

2. 프로젝트 빌드

435개의 프로젝트 각각에 대해 기본 빌드 명령을 스크립트를 통해 자동으로 적용하여 빌드를 수행한다. 빌드 성공과 실패의 판단 여부는 기본 빌드 명령 프로세스(./gradlew build)의 종료 코드로 판단한다. 빌드가 성공적으로 수행된 경우 종료 코드는 0을 나타내고, 0이 아닌 종료 코드는 빌드 실패 또는 시간 초과를 나타낸다. 빌드 스크립트가 다수 존재할 경우 프로젝트의 루트에서 명령을 실행한다. 빌드 환경은 Linux Ubuntu OS, Java SE Development Kit 8, Gradle 6.5로 구성된다.

기본 빌드 명령을 각 프로젝트에 적용한 결과는 그림 2에 나와 있다. 결과는 전체 프로젝트 435개 중 46%(200

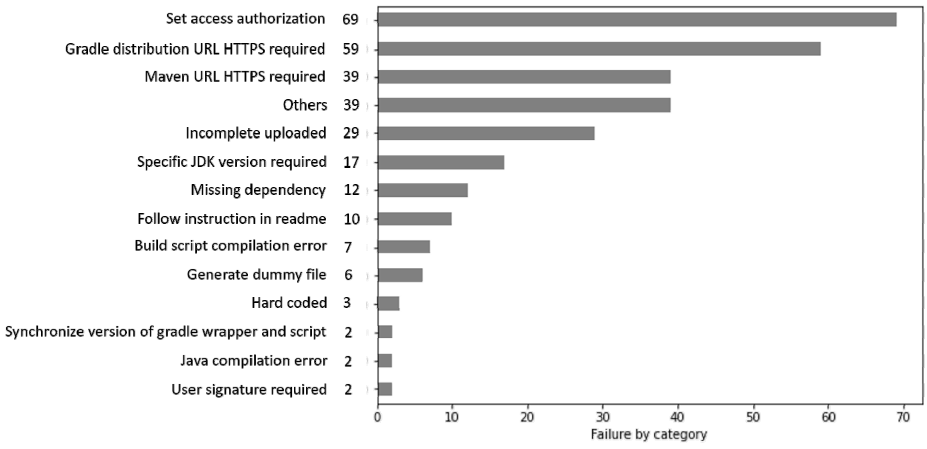


그림 3. 솔루션 범주
Fig. 3. The solution categories

개)가 빌드에 실패하고 있으며, 이는 빌드 도구가 여전히 많은 오류를 해결하지 못하고 있음을 보여준다.

3. 솔루션 범주화

빌드 오류 해결을 위한 솔루션을 찾기 위해 빌드에 실패한 200개의 프로젝트를 분석한다. 빌드 실패로 인해 직면하는 오류 메시지를 분석하고, 성공적으로 빌드가 수행 될 때까지 수동적으로 빌드 해결을 반복한다. 빌드는 순차적으로 진행되며 오류가 발생하면 프로세스가 종료되기 때문에 하나의 빌드 실패 프로젝트는 여러 빌드 오류를 가질 수 있고, 하나 이상의 솔루션을 적용해야 할 수 있다. 이 과정에서 296개의 오류 메시지를 수집하였고, 각 오류 메시지는 하나의 범주로만 분류된다. 그림 3은 빌드 오류의 솔루션 범주와 솔루션별 빌드 오류 메시지 수를 나타낸다.

우리는 솔루션 유형을 총 14가지 범주로 분류했다. 솔루션으로 범주화되기 위해서는 포함하는 빌드 오류의 수가 최소 2개 이상이어야 하며 해결해야 하는 빌드 실패의 근본적인 원인을 식별할 수 있어야 한다. 분류된 솔루션의 자세한 내용을 아래에서 설명한다.

Set access authorization: 유효한 엑세스 권한이 있는 폴더에 GRADLE_USER_HOME 환경 변수를 선언하지 않은 경우를 포함한다. 이를 해결하기 위해 올바른 환경 변수의 선언 혹은 프로젝트 폴더 접근 권한이 필요하다.

Gradle distribution URL HTTPS required: gradlew로 빌드된 프로젝트는 해당 프로젝트를 다른 이용자가 다운받을 수 있도록 url 형태로 생성되어 있다. 위 url을 통한 통신도 마찬가지로 HTTPS를 통한 암호화가 되어야 하며 관련 오류를 해결하기 위해gradle-wrapper.properties 파일에 작성된 distributionUrl의 프로토콜을 HTTPS로 변경해야 한다.

Maven URL HTTPS required: Maven 중앙 저장소는 더 이상 일반 HTTP 프로토콜을 통한 안전하지 않은 통신을 지원하지 않으며 모든 요청은 HTTPS를 통해 암호화되어야 한다. 관련 오류를 해결하기 위해 build.gradle 스크립트에 작성된 URL을 변경한다.

Incomplete uploaded: 프로젝트 빌드를 성공적으로 수행하기 위한 파일이 개발자의 실수 혹은 고의로 VCS에 업로드 되지 않은 경우를 포함한다.

Specific JDK version required: 상위 버전의 JDK로 작성된 프로젝트를 하위 버전의 JDK 환경에서 빌드를 수행할 경우를 포함한다. 오류를 해결하기 위하여 상위 버전의 JDK로 변경해준다.

Missing dependency: Maven 및 Gradle에서 종속성 파일은 중앙 저장소 또는 개발자가 지정한 저장소에 저장된다. 위는 서버가 동작하지 않거나 경로의 변경 혹은 개발자 로컬 경로에 설정된 경우를 포함한다. 오류 해결을 위해 네트워크에서 관련 종속성 파일을 찾아야 한다.

Follow instruction in readme: 프로젝트를 빌드 하기 위해 기본 빌드 명령어가 아닌 다른 명령어 혹은 추가적인 옵션을 적용해야 한다. gradle은 사용자 task를 정의할 수 있어 이를 위한 명령어를 가질 수 있다. 오류 해결을 위해 해당 저장소에 정의된 readme 파일을 확인한다.

Build script compilation error: 개발자가 build.gradle 스크립트 외부에서 선언한 변수를 스크립

트 내부에서 사용하거나 gradle 버전이 변경됨에 따라 발생하는 문법의 변경으로 인한 오류를 포함한다.

Generate dummy file: 빌드를 위해 로컬 파일(local.property)이 필요한 경우 비어있는 로컬 파일을 생성해 줌으로 빌드를 성공시킬 수 있다.

Hard coded: 홈의 경로나 파일 경로, root 폴더명 등이 하드 코딩되어 빌드에 실패한 경우를 포함한다.

Synchronize version of gradle wrapper and script : gradle-wrapper.properties는 프로젝트 배포를 위한 정보들이 작성되어 있다. 배포 버전과 빌드 버전의 차이가 오류를 발생시키므로 두 버전의 동기화가 필요하다.

Java compilation error: 프로젝트가 가진 결함을 통해 발생할 수 있는 오류를 포함한다. 프로젝트가 완전하지 않거나 변경 사항이 적용 중인 프로젝트일 수 있다.

User signature required: 개발자 아이디나 비밀번호 혹은 개발 토큰과 같이 사용자가 직접 생성하고 입력해야 하는 경우를 포함한다.

Others: 빌드 실패를 발생시키는 오류의 근본적인 원인이 확실하지 않거나, 원인이 식별되어 솔루션이 적용되었으나 자주 발생하지 않아 솔루션이 범주화되지 않은 경우를 포함한다.

위 범주에서 사용자에게 직접 인증키를 입력받거나 공유된 코드가 불완전한 경우를 포함해서 모든 오류에 대한 직접적인 솔루션을 제시할 수 없는 오류도 존재한다. 본 연구에서는 오류를 발생시키는 원인을 식별하는 것 또한 잠재적으로 오류를 해결할 수 있는 여지를 준다고 판단하여 솔루션으로 정의하고 범주화를 진행했다.

4. 기계 학습

빌드 오류 메시지와 관련된 솔루션을 추천해주기 위해 먼저 자연어에 대한 전처리가 실행된다. 빌드 오류 메시지에는 자연어뿐만 아니라 소스 코드 및 환경 경로 같은 노이즈를 포함해 모델이 학습하기에 부적절하다. 따라서 솔루션 추천을 위한 모델이 잘 분류해 낼 수 있도록 다음과 같은 추가적인 처리가 이루어진다.

- 토큰화: 오류 메시지에 포함되는 자연어는 공백을 통해 단어 단위로 분할되며 파일 경로 및 패키지 경로는 콤마로 구분되어 분할된다.
- 불용어 제거: "a", "the"와 같은 불용어는 솔루션을 분류하는데 의미가 없다. 따라서, 이러한 불용어는 WordNet 영어 불용어 목록에 따라 제거된다.
- 어간 추출: 단어는 기본 형태인 어간으로 변형된다. 예를 들어, "Starting"은 start로 변경된다.
- 표제어 추출: 단어의 기본형으로 분석될 수 있도록 단어의 변형된 형태를 그룹화한다. 예를 들어, "has"는 "have"로 변경된다.
- 특수문자 제거: 영어가 아닌 문자의 ASCII 코드는 65-90 그리고 97-122의 범위를 벗어난다. 따라서 정규 표현식을 통해 메시지의 특수문자를 제거한다.

결과적으로 위 프로세스는 솔루션 추천을 위한 단어 모음을 반환하고 기계학습 모델의 입력으로 사용한다. 본 연구에서는 기계학습 분야에서 분류 모델로 사용되는 대표적인 모델 4가지를 사용한다. SVM, Logistic Regression, Random Forest, XGBoost를 사용하여 솔루션 추천 모델을 구축하고 모델별 성능을 비교한다.

Ⅳ. 실험 및 결과

본 연구에서 빌드 실패 프로젝트의 해결을 위해 솔루션 추천 모델을 구축하고 분류 모델별 성능을 비교한다. 실험에 사용할 데이터 셋은 빌드 실패를 통해 수집된 296개의 빌드 오류 메시지와 14개의 솔루션 범주로 구성된다. 데이터 수집 방법 및 구성은 3절에서 설명되었다.

검증으로는 분류 모델 성능 측정의 통계적 신뢰도를 높이기 위해서 resampling 방법인 교차검증을 사용한다. 본 실험에서는 10-fold 교차 검증을 시행하였다. 솔루션 추천 모델 학습을 위해 수집된 빌드 오류 메시지 셋의 80%를 사용하고 검증을 위해 20%를 사용해 분류 모델을 평가하였다. 이때, 모든 클래스에 대해 데이터 비율을 유지한 채 분할한다. 성능 측정 지표로는 혼동행렬의 정확도(Accuracy)를 사용하였으며 다음과 같이 정의한다.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (1)$$

본 연구에서는 기계학습 분야에서 분류 모델로 사용되는 대표적인 4가지 알고리즘을 사용해 실험했다. BoW(Bag of Words), TF-IDF, n-gram, Word2Vec를 통해 오류 메시지를 벡터화 하고 입력값으로 사용하여 분류 모델간의 성능을 비교한다. Word2Vec는 Google 뉴스에서 300만 개의 단어를 사전 학습하였다.

표 1. 빌드 오류에 대해 서로 다른 모델을 사용한 솔루션 추천 정확도
Table 1. Accuracy of solution recommendations using different models for build errors

Model	Method				
	BoW	TFIDF	BoW+TFIDF	BoW+2-gram	Word2Vec
SVM	0.8579	0.8616	0.8680	0.8647	0.8407
Logistic Regression	0.8818	0.8378	0.8200	0.8748	0.7350
Random Forest	0.8918	0.8645	0.8644	0.8952	0.8695
XGBoost	0.8610	0.8305	0.8680	0.8557	0.7736

표 1은 솔루션 추천에 대한 정확도 측정값을 보여준다. 4가지 알고리즘 중 가장 높은 정확도 측정값에 굵게 표시된다. 실험 결과로 Bag of Words와 2-gram을 함께 사용해 오류 메시지를 벡터화하여 학습한 Random Forest가 0.8952의 성능으로 가장 높은 정확도를 보인다. TFIDF는 모델을 학습시키기 위한 효과적인 벡터화 방법이나 소스 코드가 포함되어있는 오류 메시지에 적용하는 것은 전반적으로 낮은 성능을 보여준다. Word2Vec 경우 사전 학습된 모델을 사용하여 추가적인 학습을 진행해 높은 성능을 기대하였으나 가장 낮은 분류 성능을 보여주고 있다. 이는 사전 학습된 Google 뉴스와 오류 메시지간 사용하는 단어에 차이가 있기 때문으로 판단된다.

V. 위협 요소

내적 타당성에 대한 위협은 데이터 셋의 수집, 빌드 오류 해결을 위한 솔루션의 범주화, 접근 방식의 구현이다. 모든 실험은 저자 한 사람에 의해 수행되었으나 *공개된 소스 코드와 데이터를 통해 본 연구에서 수행한 연구를 동일하게 재현 가능하다. 또한, 솔루션을 범주화하기 위해 오류에 대해 논의하고 빌드 로그를 조사했으며 범주에 대한 도움을 구하기 위해 웹 포럼을 검색했다. 오류의 근본적인 원인이나 해결책이 확실하지 않을 경우 솔루션을 별도로 범주화하였다.

외적 타당성에 대한 위협은 본 실험의 결과가 제시된 기준에 의한 프로젝트에만 적용될 수 있다. Maven 및 Ant와 같은 다른 빌드 도구의 실패와 다른 프로그래밍 언어는 향후 추가 실험을 통해 유효성을 검사해야 한다. 하지만 본 연구에서 제시한 접근 방법은 언어 및 빌드 도구에 구애받지 않고 확장 가능하다.

* https://github.com/minqukanq/solutions-for-resolving-build-errors

VI. 결 론

본 연구에서는 프로젝트를 구축하는 과정에서 발생하는 빌드 실패를 자동으로 해결할 수 있는 솔루션 추천 연구를 수행하였다. 빌드 오류를 분석하고 이를 해결할 수 있는 솔루션을 범주화하여 제시했다. 특히 435개의 Java 프로젝트에서 200개의 빌드 실패를 확인하였으며, 여기에서 추출된 296개의 빌드 오류 중 89%(264개)에 솔루션을 추천하였다. 결과적으로 소프트웨어 사용자가 직면한 빌드 오류를 해결하는데 실질적인 해결책을 제공하였고, 빌드 도구의 발전 및 자동화에 기여할 수 있을 것이라 기대한다.

향후 연구로는 Maven 및 Ant와 같은 다른 빌드 도구까지 확장하여 보다 많은 데이터 셋을 확보하고, 빌드 실패를 자동으로 해결하는 도구를 개발할 예정이다.

References

[1] Sulír, M., & Porubän, J, “A quantitative study of java software buildability”, Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools, pp. 17-25, November 2016.
DOI:https://doi.org/10.1145/3001878.3001882

[2] Müllerburg, Monika AF, “The role of debugging within software engineering environments”, Proceedings of the symposium on High-level debugging, pp. 81-90, August 1983.
DOI:https://doi.org/10.1145/1006147.1006165

[3] Hailpern, Brent, and Padmanabhan Santhanam, “Software debugging, testing, and verification”, IBM Systems Journal, Vol. 41, No. 1, pp. 4-12, January 2002.
DOI:https://doi.org/10.1147/sj.411.0004

[4] Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., & Parnin, C, “Do developers read compiler error messages?”, International Conference on Software Engineering (ICSE), pp. 575-585, May

2017.
DOI:https://doi.org/10.1109/ICSE.2017.59

[5] W. J. Lee, I. W. Choi, "A Tool to Support Efficient Development of Node Software for Various Operating System Platforms in Sensor Network Environment", Journal of the Korea Academia-Industrial cooperation Society, Vol. 15, No. 7, pp 4536-4544, 2014.
DOI : 10.14801/kiitr.2013.11.2.119

[6] J. Y. Kim, K. K. Kwon, S. I. Lee, and K. S. Ahn, "Implementation of IDE for OSEK/VDX OS Extended Real-time Processing and Protection Functions", The Journal of Korean Institute of Information Technology, Vol. 11, No. 2, pp 119-126, February 2013.
DOI : 10.14801/kiitr.2013.11.2.119

[7] McIntosh, S., Adams, B., Nguyen, T. H., Kamei, Y., & Hassan, A. E, "An empirical study of build maintenance effort", IEEE International Conference on Software Engineering, pp. 141-150, May 2011.
DOI:https://doi.org/10.1145/1985793.1985813

[8] Seo, H., Sadowski, C., Elbaum, S., Aftandilian, E., & Bowdidge, R, "Programmers' build errors: a case study (at google)", In Proceedings of the 36th International Conference on Software Engineering, pp. 724-734, May 2014.
DOI:https://doi.org/10.1145/2568225.2568255

[9] Hassan, F., Mostafa, S., Lam, E. S., & Wang, X, "Automatic building of java projects in software repositories: A study on feasibility and challenges", ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 38-47, November 2017.
DOI:https://doi.org/10.1109/ESEM.2017.11

[10] Hassan, F., & Wang, X, "Hirebuild: An automatic approach to history-driven repair of build scripts", In 2018 IEEE/ACM 40th International Conference on Software Engineering, pp. 1078-1089, May 2018.
DOI:https://doi.org/10.1145/3180155.3180181

저 자 소 개

강 민 구 (준회원)

- 2020.2~: 전북대학교 석사 재학
- 2020: 전북대학교 학사
- Email: 201314090@jbnu.ac.kr

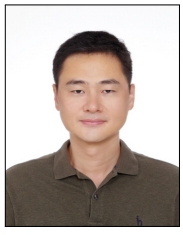
김 태 영(준회원)

- 2018.2~: 전북대학교 박사 재학
- 2018: 전북대학교 공학석사
- 2016: 전북대학교 학사
- Email: rlaxodud1200@jbnu.ac.kr

김 순 태 (정회원)

- 2014~: 전북대학교 소프트웨어공학과 교수
- 2010: 서강대학교 공학박사
- 2007: 서강대학교 공학석사
- Email: stkim@jbnu.ac.kr

류 덕 산 (정회원)



- 2018~: 전북대학교 소프트웨어공학과 조교수
- 2016: KAIST 공학박사
- 2012: KAIST 및 CMU 공학석사
- Email: duksan.ryu@jbnu.ac.kr

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임. (NO.2020R1F1A1072039)